CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2017;e4231.:1–15
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/CPE.4231

A scalable synthetic traffic model of Graph500 for computer networks analysis

Pablo Fuentes[1*] Mariano Benito[1], Enrique Vallejo[1], José Luis Bosque[1], Ramón Beivide[1], Andreea Anghel[2], Germán Rodríguez[3], Mitch Gusat[2], Cyriel Minkenberg[3] and Mateo Valero[45]

[1]*University of Cantabria, Dept. Computer Science and Electronics, Avda. Los Castros s/n 39005, Cantabria, Spain.*
[2]*IBM Zurich Research Laboratory, Säumerstrasse 4, 8803 Rüschlikon, Switzerland.*
[3]*Rockley Photonics Inc, Switzerland.*
[4]*Universitat Politecnica de Catalunya, Barcelona, Spain*
[5]*Barcelona Supercomputing Center, Barcelona, Spain.*

SUMMARY

The Graph500 benchmark attempts to steer the design of High-Performance Computing systems to maximize the performance under memory-constricted application workloads. A realistic simulation of such benchmarks for architectural research is challenging due to size and detail limitations. By contrast, synthetic traffic workloads constitute one of the least resource-consuming methods to evaluate the performance.
In this work, we provide a simulation tool for network architects that need to evaluate the suitability of their interconnect for BigData applications. Our development is a low computation- and memory-demanding synthetic traffic model that emulates the behavior of the Graph500 communications, and is publicly available in an open-source network simulator. The characterization of network traffic is inferred from a profile of several executions of the benchmark with different input parameters.
We verify the validity of the equations in our model against an execution of the benchmark with a different set of parameters. Furthermore, we identify the impact of the node computation capabilities and network characteristics in the execution time of the model in a Dragonfly network.

Received: 1 November 2016
Accepted: 2 June 2017

KEY WORDS: scalable simulation tools; system interconnect; network evaluation; Graph500 benchmark

1. INTRODUCTION

BigData applications have become ubiquitous and gather the interest of system architects and designers. Many of these workloads are more memory- and IO-bounded and place a higher stress on the network system than traditional High-Performance Computing (HPC) applications. This behavior makes the nature of their communications of high relevance for network architects, who require new tools for an adequate development of future systems.

One of the biggest challenges of BigData applications is to structure data to rapidly find relations and establish associations. A typical approach is to organize data into graphs and explore them following a tree. There exist multiple strategies to obtain a tree given a graph; Breadth-First Search (BFS) is one of the most known and employed among them.

*Correspondence to: Pablo Fuentes, Dept. Computer Science and Electronics, Facultad de Ciencias, Universidad de Cantabria, Avda. Los Castros s/n 39005, Cantabria, Spain. E-mail: pablo.fuentes@unican.es

The Graph500 benchmark [1] [2] appeared in 2010 with the aim of influencing the design of new systems, so they better adjust to the memory- and IO-bounded requirements of data intensive applications. Based on the execution of a BFS within a graph, it is currently one of the most known BigData-focused benchmarks. Therefore, it is strongly useful for the evaluation and design of parallel computers, especially their memory and network subsystems.

The objective of this work is to provide computer network architects with a simple and scalable tool to evaluate the impact of the system interconnect on the execution of a BigData workload. To achieve it, we review, extend and implement a synthetic traffic model of the communications of the Graph500 benchmark. A publicly available implementation in an open-source network simulator is provided, and can be handily adapted to other simulators. Our model replicates the staged structure of the benchmark with large batches of uniformly distributed point-to-point messages ending in a collective all-reduce operation. The number of messages for each stage is defined through a Gaussian distribution whose mean and standard deviation are a function of the input parameters of Graph500. The equations for the mean and standard deviation are obtained empirically from a characterization of several benchmark executions with different input parameters.

In summary, the main contributions of this paper are:

- We extend a synthetic traffic model that follows the behavior of the Graph500 benchmark, introduced in a previous work in [3]. Our model predicts the volume of point-to-point communications in each stage of the model.
- We implement our model in an open-source network simulator [4]. This model achieves good scalability both in size of the simulated network and maximum graph size explored in the benchmark, without extensive computational or memory requirements.
- We validate our model through a comparison against empirical values from the execution of the Graph500 benchmark, and present a series of results from the execution of our model, considering a Dragonfly network. We analyze these results and link them to the characteristics of the workload.

The remainder of the paper is organized as follows: first, we perform a brief analysis of the Graph500 benchmark and the behavior of its communications. Then, we present our synthetic traffic model. Next, we give a comprehensive description of the implementation of the model in a network simulator. A validation of the model follows, crosschecking the validity of the equations in the model and presenting results from the implementation in a network simulator. Finally, we present related work and the conclusions of the paper.

2. ANALYSIS OF THE BENCHMARK COMMUNICATIONS

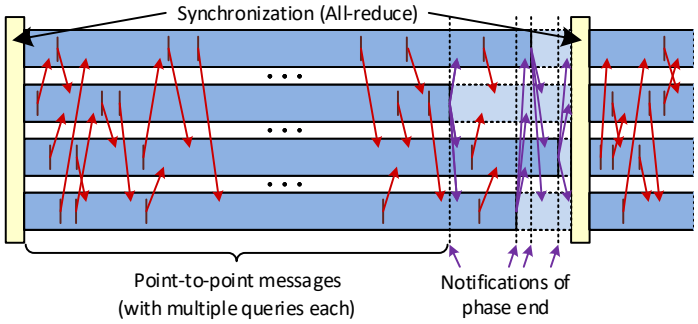
This section extends the analysis of the communications of the Graph500 introduced in [3]. The benchmark consists of two kernels: a graph generator and a BFS performed for a number of randomly selected tree vertices across the graph. The benchmark receives two input parameters defining the size of the graph (*scale* and *edgefactor*) and returns the achieved system performance along the execution of the BFS. Four implementations of the BFS are provided by default; in this work we focus on the most-scalable *simple* implementation that splits the graph among the processes without replicating data. A thorough description of the implementations can be found in [5] and [6].

Communications along the benchmark execution are tied to the levels of the tree obtained from the BFS execution. In each level, each process searches the neighbors of the current frontier of the tree. Those vertices are marked as part of the next frontier, so they are visited in the following level. A query is generated when a given neighbor vertex is allocated to a different process. This query corresponds to a petition to determine if the vertex has been already visited. These queries translate into point-to-point messages. From a network perspective, each tree level consists of a batch of point-to-point messages followed by a final notification to all other processes, to signal the end of the level within the process.

Figure 1 gives some understanding of the communications during a BFS execution. Figure 1a portrays an outline of the communications within each phase of the BFS, depicting the dispatch

DOI: 10.1002/CPE.4231

(a) Outline of the communications in each BFS phase.



(b) Trace of an actual execution with 16 processes. Scale $s = 22$ and edgefactor $f_e = 16$.

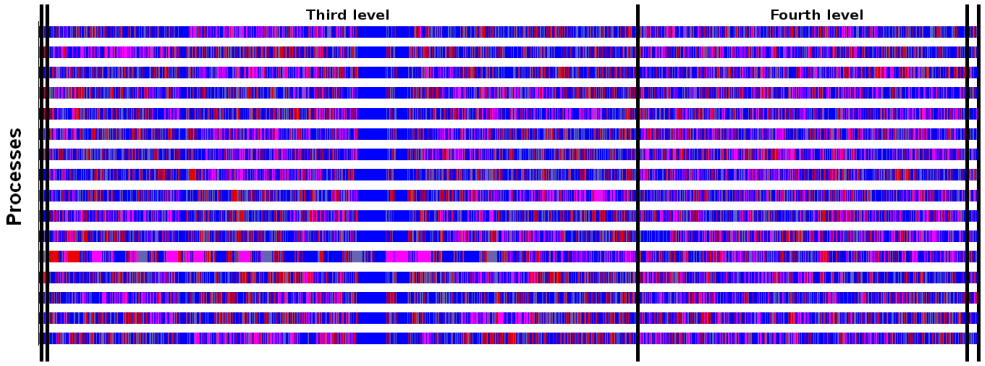


Figure 1. Temporal evolution of the communications of the BFS.

of messages through arrow lines. Processes communicate the end of message dispatching at the current phase trough a phase end notification. The end of each phase is marked by a synchronization point through an all-reduce, which comprises a reduction and a broadcast. This collective operation ensures both the distribution of the number of new visited vertices in the stage, and a synchronous beginning of the next phase. Between the dispatch of the notification of phase end and the beginning of the all-reduce (shown in light blue), processes only perform the associated computation of the incoming messages. The length of this block is marked by the slowest process to enter the all-reduce.

Figure 1b presents a trace of a BFS taken from the execution of the Graph500 benchmark over a graph of *scale* 22 and *edgefactor* 16, employing 16 processes. The blue blocks in the trace represent computation, and pink blocks correspond to the dispatch of a message. Black bars point the all-reduce collectives. It can be appreciated that communications are interspersed with computation along the execution, with message shipment uniformly distributed across the execution.

The amount of messages per level is distributed evenly between all processes, but it varies significantly between tree levels. Two big levels typically represent around 80% of the total. The impact of the point-to-point messages to notify the end of a tree level is negligible [7]. The likewise sporadic all-reduce is important only because it synchronizes the generation of new messages. Therefore, from a network usage perspective, the major communications are the point-to-point exchanges during each tree level. These communications are highly homogeneous spatially due to the even distribution of the graph across processes.

Graph500 exploits message aggregation to reduce network traffic, with every message grouping multiple queries up to a value named *coalescing size*. Each query corresponds to a petition to explore the vertex linked through a given edge, so the number of queries exchanged between two processes equals the number of explored edges attached to vertices hosted in the two processes for a given tree level. The even distribution of the graph across processes translates into an even distribution of

DOI: 10.1002/CPE.4231

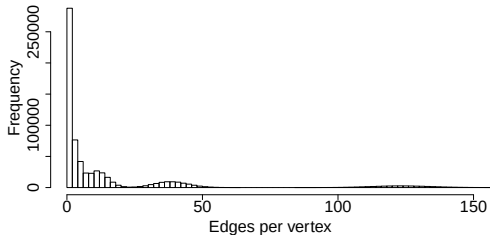


Figure 2. Histogram of the vertex degree, truncated. Graph scale is $s = 17$, edgefactor is $f_e = 16$.

the total number of explored edges per tree level among all the possible source-destination process tuples. The graph is undirected, forcing all edges to be traversed twice - one per each way.

Graphs in the benchmark are generated through a Kronecker matrix product similar to the Recursive MATrix (R-MAT) scale-free graph generation algorithm [8]. Graphs generated by R-MAT emulate the behavior of small-world phenomena, where a small fraction of the nodes (or *vertices*) have a significantly large number of direct connections (or *edges*) with other nodes, and a large proportion of the vertices have a low number of edges (or *vertex degree*) [9]. Figure 2 displays a histogram of the vertex degree measured for a graph generated in the benchmark. A reduced set of sparse higher degree values (up to around 30,000 edges) has been truncated from the figure for ease of visualization. It can be appreciated that the distribution is heavily condensed in low degree values, with a trend of tails that favor some high degree values over others. Previous works from Chakrabarti and Faloutsos [8], Leskovec *et al.* [10] and Kim and Leskovec [11] characterize the distribution of the vertex degree in such graphs as power-law or lognormal, which is the distribution employed in our model.

3. GRAPH500 NETWORK TRAFFIC MODEL

This section presents the model, which is extended from [3] by considering the variance of the number of messages per level to support variability in the execution. Our model consists of a traffic generator structured in multiple stages corresponding to the consecutive tree levels. For each stage, a given number of point-to-point messages is dispatched from every process; these messages are uniformly distributed in time and across destinations (following the analysis of the benchmark communications in [7]) and are succeeded by end-of-phase signals and a synchronization point.

As explained in the previous section, the most critical part of the execution for any non-trivial graph size will be the point-to-point communications, since the messages will significantly outnumber those during the all-reduce. This section provides a comprehensive insight of the point-to-point communications and its temporal and spatial distribution.

Table I presents a summary of the abbreviations that are employed in the model equations, with a brief description of each parameter. The *scale*, *edgefactor* and *coalescing size* are input parameters of the benchmark, and the number of nodes n is directly tied to the benchmark execution; the rest of the variables in the table are internally employed.

Point-to-point message generation rate depends fundamentally on the node computation capabilities, and we introduce it as an input parameter to the model that is constant across any execution. Any given set of nodes can be tuned to under- or out-perform the rest of the system, in order to model heterogenous systems. Message destination is selected randomly among all the other processes in the application, following the uniform distribution of the vertices observed in the execution of the benchmark [7]. In every level the number of messages sent for each process is determined following Equations 1- 3.

Prepared using cpeauth.cls
DOI: 10.1002/CPE.4231

Table I. List of abbreviations employed in the equations.

Abbr.	Parameter	Description
s	Scale	Base 2 logarithm of the number of vertices in the graph.
f_e	Edgefactor	Half of the average vertex degree.
c_s	Coalescing size	Amount of explored edges aggregated per message.
n	Number of nodes	Number of nodes (processes) employed in the benchmark execution.
n'	Number of destination nodes	Number of nodes (processes) which actually receive a message in a given level.
q	Number of queries per process	Number of queries sent from each process per tree level.
m	Messages per process	Number of messages sent from each process per tree level.
E_l	Edges per tree level	Total number of edges explored within each stage of the BFS.
d_r	Degree of the root	Number of edges connected to the root vertex.
l	Tree level	Stage of the tree in the BFS execution, starting at 0.

$$q = \frac{E_l}{n} \cdot \frac{n-1}{n} \tag{1}$$

$$n' = n \cdot \left(1 - \left(1 - \frac{1}{n}\right)^q\right) \tag{2}$$

$$m = \left\lceil \frac{q}{c_s} \cdot \frac{1}{n'} \right\rceil \cdot n' \tag{3}$$

Equation 1 refers the number of queries as a function of the number of edges explored per node and tree level, minus those edges that are explored within the node and do not span any network communication. The number of explored edges per tree level is evaluated in Section 3.1.

Equation 2 defines the actual number of destinations (n') as a Bernoulli experiment [12] with as many trials as queries sent and a uniform probability of selecting a given node as destination for a query. If $q \to \infty$, the number of effective destination equals the number of nodes in the network, n.

The number of messages given in 3 equals the quotient of the number of queries sent per tuple of (source, destination) processes times the number of destination processes, and the message aggregation. Due to message aggregation, each message can carry up to as many queries as the value of the *coalescing size* (c_s). The number of messages between every pair of nodes is rounded up to account for those messages carrying less than c_s queries.

3.1. Characterization of the mean and standard deviation of the number of edges per tree level

The total number of explored edges across the whole BFS execution is almost constant for a pair of given *scale* and *edgefactor* parameter values, but its allocation among tree levels varies heavily. Figure 3a depicts the histogram of the number of new edges traversed in a graph during the third tree level ($l = 2$), which gathers most of the point-to-point communications of the whole execution as discussed in the previous section. This histogram has been obtained by running a BFS for every root in the graph and does not seem to follow any clear distribution, because simply averaging multiple executions with the same input parameters masks the actual behavior.

Our approach is to determine the number of explored edges per tree level as a function of the degree of the root vertex, which heavily influences the distribution. Figures 3b-3d display the average distribution of the number of explored edges during the third tree level for the same graph in Figure 3a, with three different ranges of root degree: roots with only one neighbor (Figure 3b), roots with 10 to 20 neighbors (Figure 3c) and roots with a high root degree of 10,000 neighbors or more (Figure 3d). For each range of the root degree there is only one predominant peak, as opposed to the histogram for all roots shown in Figure 3a where there are multiple peaks of similar impact. This heavy dependence on the root degree comes from the dynamic evolution of the benchmark: roots with low vertex degree originate a low amount of communications at first tree levels, shifting the biggest part of the graph traversal to higher levels, whereas roots with high degree rapidly explore the majority of the graph and present low (or non-existent) communication at higher levels.

Prepared using cpeauth.cls
DOI: 10.1002/CPE.4231

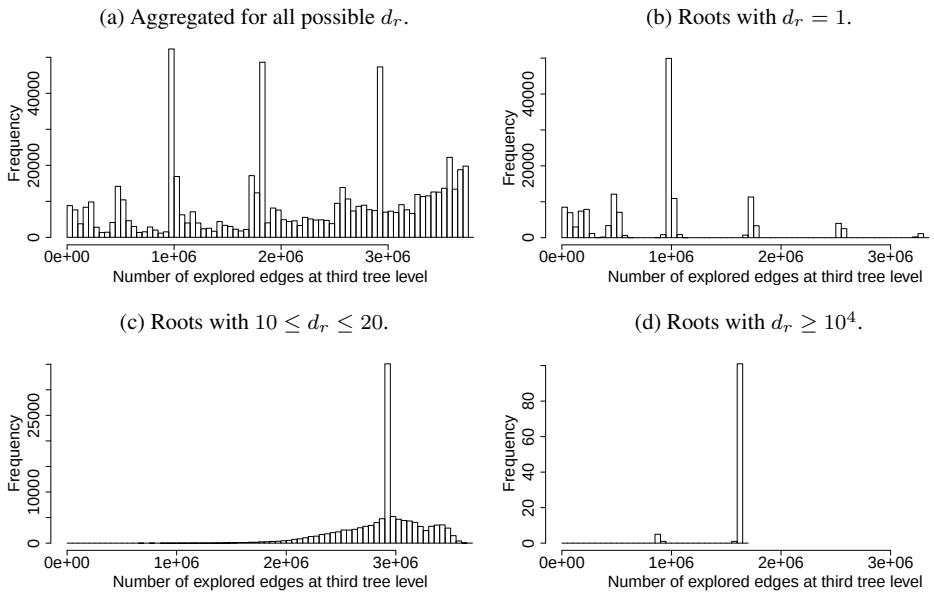


Figure 3. Histogram of the number of explored edges in the third tree level, with different root degree d_r. Graph with scale $s = 17$ and edgefactor $f_e = 16$.

We model the number of edges explored per tree level and process through a Gaussian distribution. This method has the benefit of adjusting reasonably to the observed behavior while remaining computationally low demanding.

The equations for the mean and standard deviation of the Gaussian distribution have been obtained through a statistical analysis of the average number of explored edges from a set of executions of the benchmark for different input parameters. The Gaussian distribution does not apply for the first tree level, where the number of explored edges is trivially determined as the root degree.

This analysis has been conducted by establishing a linear combination that fits the observed values through a model fitting tool based on the function described by Chambers in [13]. The coefficients of said linear combinations have then been generalized to follow the variations with the input parameters. The measurements are oblivious to the infrastructure employed, so they can be extrapolated to any other system.

Our model characterizes the mean and standard deviation of the number of visited edges per tree level as a function of the root degree and the graph parameters (*scale* and *edgefactor*).

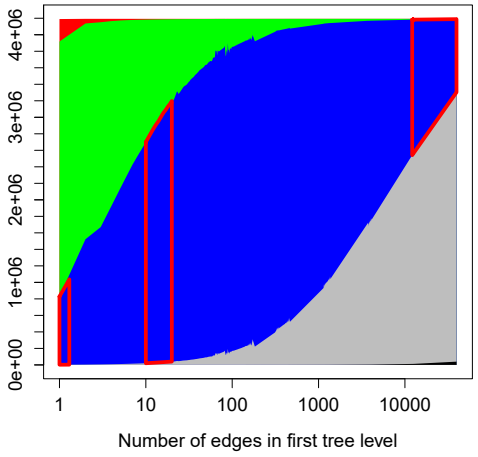
Figures 4 and 5 depict the mean and standard deviation of the number of edges upon the root degree, broken down per tree level. X-axis is displayed in logarithmic scale. Note that Y-axis in Figure 4b is also in logarithmic scale. Results in Figure 4 correspond to the same data used for the histograms in Figure 3. The three blocks circled in red correspond with the average number of explored edges in the third tree level whose distribution was presented in Figures 3b-3d. Some values in Figure 4 are interpolated, as not all the vertex degrees are present in a graph. This translates into the gaps in the curves of the standard deviation (Figure 5). The aggregated amount of edges remains almost constant, since the size of the graph is independent of the vertex selected as root (and consequently the amount of edges to traverse during the BFS will be similar). However, the distribution of those edges among the tree levels varies significantly, further confirming our motivation to relate the communications to the vertex degree at the tree root. In the following subsection we will present an equation linking the number of edges per tree level to the root degree.

3.2. Number of explored edges per tree level

Our model estimates the evolution of the number of explored edges per process for each tree level through a Gaussian distribution. The first tree level is an exception to this, with the number of

Prepared using cpeauth.cls
DOI: 10.1002/CPE.4231

(a) Stacked values.



(b) Curves.

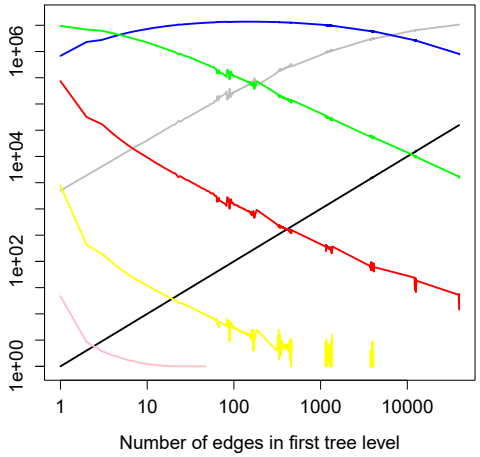


Figure 4. Number of explored edges per root degree, broken down per tree level. Note that the Y-axis is in logarithmic scale for the right figure.

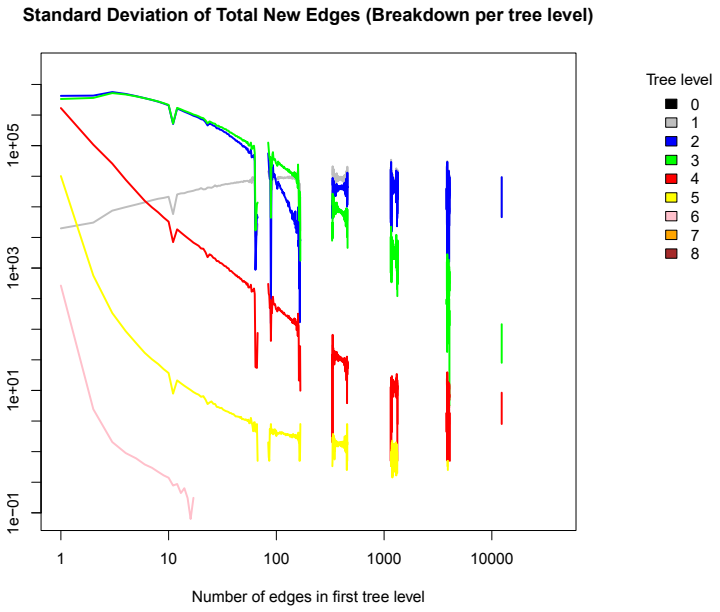


Figure 5. Standard deviation of the number of explored edges per root degree for each tree level.

explored edges being defined directly as the degree of the root vertex (and originating messages only at one process, the one hosting the root).

The Gaussian distribution is characterized by a mean and standard deviation that are a function of the root degree d_r, the tree level l and the input parameters of the benchmark: the *scale* of the graph (s) and the edgefactor (f_e). We consider a notation for the tree level l that spans from $l = 0$.

The evolution of the mean per tree level (shown in Figure 4b) is estimated through a polynomial of degree 2 as the one described in Equation 4. We have approximated the A, B and C factors in each tree level to fit the observed values for several combinations of *scales* $s = 16, 17, 18, 19, 20, 23, 25$

Prepared using cpeauth.cls
DOI: 10.1002/CPE.4231

and *edgefactors* $f_e = 16, 20, 32, 45, 64$. For each explored combination, we have run a BFS to compute a tree for every vertex in the graph, and measured the average number of explored edges per level for each root vertex degree.

We then have obtained an expression that fits the evolution of each of the parameters in Equation 4 with the *scale* and *edgefactor* parameters as well as the tree level, accomplishing a reasonable prediction for larger scale and edgefactor values. This expression is presented in Equations 5 - 7. The mean of the Gaussian distribution is truncated when the number of edges explored through the whole execution reaches the limit of twice the number of edges in the graph. Equations 5 - 7 are an update to a previous version of this work in [3] to prevent anomalous behaviors for large scale values ($s \geq 30$).

$$\ln\left(\overline{E_l}\right) \approx A \cdot \ln^2(d_r) + B \cdot \ln(d_r) + C, \qquad l \geq 1 \tag{4}$$

$$A = -0.133 + 0.0046 \cdot s + e^{0.01257 \cdot f_e - 0.1829 \cdot s - e^{1.75 - 0.7 \cdot l}} \tag{5}$$

$$B = 2 - l \cdot (0.91 + 0.002 \cdot f_e - 0.012 \cdot s) \tag{6}$$

$$C = e^{1 + (1 + 0.004 \cdot f_e) \cdot (-0.81 + 0.8411 \cdot \ln(s)) \cdot e^{-\frac{(l-2.8)^2}{30}}} \tag{7}$$

A similar analysis is performed for the standard deviation, although in this case the approximation function changes significantly between tree levels. Lower levels can be described through an exponential of a polynomial like the one used for the average, whereas higher levels fit better into an exponential of an inverse function. Interestingly, the second and third tree levels present a dual nature; in the second level, there are two zones with different trends, and the second zone follows the curve from the first level. Something similar happens for the third tree level, in which the first points equal those of the fourth level and the remaining match the values of the second level. In both cases, this is equivalent to taking the maximum between the crossing curves (this third case is not spotted in this figure, but occurs with higher scale sizes).

Equation 16 refers said model, where the functions $f_1(l, d_r)$ and $f_2(l, d_r)$ are determined by the formulas in Equations 8 - 15. These equations reflect a high divergence for roots with low degree, which have a much larger number of samples in the graph. High degree roots are much sparser, reducing the deviation between samples.

$$f_1(l, d_r) = D \cdot \ln(d_r)^2 + F \cdot \ln(d_r) + G \tag{8}$$

$$f_2(l, d_r) = \frac{K}{{d_r}^H} + J \tag{9}$$

$$D = 0.002 \cdot s - 0.14 - (0.015 \cdot s - 0.285)(0.56 + 0.033 \cdot f_e)(l - 1) \tag{10}$$

$$F = 0.97 - (l - 1)(4.438 - 0.168 \cdot s)(0.83 + 0.01 \cdot f_e) \tag{11}$$

$$G = ((2.1 + l)(5.35 + 0.093 \cdot s) - 13.625)\left(1.23 - \frac{2.9 + 0.69 \cdot l}{f_e}\right) \tag{12}$$

$$H = 2^l \cdot (0.011 + 0.00012 \cdot f_e) \cdot e^{1.7 - \frac{s}{10}} \tag{13}$$

$$J = 1 - e^{(1 - 0.012 \cdot f_e) \cdot (9.55 - 1.3 \cdot l - s \cdot (0.6 - 0.1 \cdot l))} \tag{14}$$

$$K = 13 \cdot (0.062 \cdot s - 0.046) \cdot (2 - 0.24 \cdot l) - J \tag{15}$$

$$\ln(\sigma_{E_l}) \approx \begin{cases} f_1(l, d_r) & l = 1, \\ \max(f_1(1, d_r), f_1(2, d_r)) & l = 2, \\ \max(f_1(2, d_r), f_2(4, d_r)) & l = 3, \\ f_2(l, d_r) & l > 3. \end{cases} \tag{16}$$

DOI: 10.1002/CPE.4231

Table II. List of abbreviations employed in the model implementation.

Abbr.	Parameter	Description
p_s	Packet size	Size of the packet, in phits. A phit is the number of bytes sent per simulation cycle.
p_{inj}	Injection probability	Probability of injecting a phit in a given cycle.
t_q	Query time	Number of cycles of computation associated with processing a query.
t_c	Cycle time	Duration of a simulation cycle, in seconds.

Table III. List of query computation time for different node architectures.

Node	t_q
Altamira supercomputer - IBM iDataplex dx360m4, Intel Xeon E5-2670 @2.6GHz, 64GB RAM @1.6GHz	1.5ns
Intel Core i5-5200U @2.2GHz, 8 GB RAM @1.6GHz	2.25ns
Intel Xeon E5-2620 @2GHz	2.4ns
Mont-Blanc prototype [14] - Samsung Exynos 5, ARM Cortex A15 @ 1.7GHz	15ns

4. IMPLEMENTATION OF THE MODEL IN A NETWORK SIMULATOR

We have implemented our model in the FOGSim network simulator [4] in order to evaluate the impact of the Graph500 traffic workload on system networks. Our implementation receives as input parameters the scale of the graph (s), the edgefactor (f_e) and the coalescing size (c_s) in order to determine the amount of point-to-point messages that will be transmitted during the execution. Additionally, it requires a value for the time invested in the associated computation of an incoming query. This query computation time t_q is also used to compute the rate at which the point-to-point messages will be injected into the network. Table II presents a list of abbreviations used in the implementation of the model.

As outcome, the simulator reports the time invested in completing the execution of one Breadth-First Search, as well as various sets of network statistics about the execution. It does not consider the invested time in the generation of the graph, since that part of the benchmark execution is overlooked for the performance figures of the benchmark.

Since the simulator is cycle-accurate, the time invested in the computation of a query must be input in cycles. The injection probability of the point-to-point messages is expressed as a percentage of the maximum traffic load that can be delivered by the network. For a given node architecture, Equation 17 determines the corresponding injection probability at the simulator as a function of the query execution time and the length of a simulation cycle.

$$p_{inj} = \frac{p_s \cdot t_c}{t_q \cdot c_s} \tag{17}$$

Query computation time is highly related to the performance of the hypothetical machine being evaluated. In order to ease model usability, Table III provides the per-query computation times measured under different node architectures, including a conventional HPC cluster and a novel ARM-based cluster prototype. This serves as a reference to estimate the computation capabilities for a given architecture; should a more precise value be desired, t_q can be quantified running an instrumentalized version of the Graph500 benchmark.

The difference between architectures is not very significant in absolute terms; query processing is not computing-intensive and is mainly bounded by memory accesses. The best performer is Altamira, an HPC cluster located at the University of Cantabria. However, a laptop chip such as the Core i5 performs only a 30% slower. The highest computation time corresponds to the Mont-Blanc prototype, an ARM low-consumption cluster. This prototype has been built as part of Mont-Blanc [14], an EU-funded program that targets energy-efficient computation through large ARM-based systems. Being based on low-power processors, a high execution time is reasonably expected, and constitutes a lower bound for the execution time in our set of architectures.

Figure 6 portrays a flowchart with the states followed by the Graph500 synthetic traffic generator. Simulation starts by determining the number of point-to-point communications that will be delivered within the current stage or tree level. Each node of the network corresponds to a process in the

DOI: 10.1002/CPE.4231

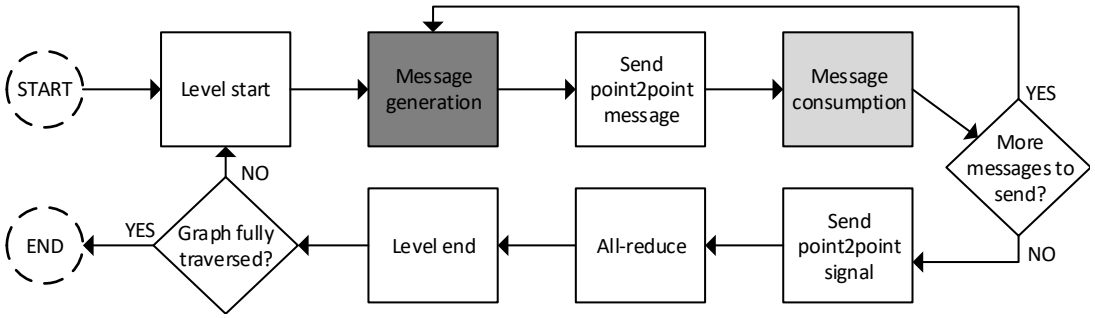


Figure 6. Flowchart describing the behavior of the Graph500 simulation model.

benchmark; for the first stage, one node is selected as host of the root vertex and begins the generation and dispatch of the point-to-point messages, being the lone sender for the whole level.

The amount of messages is calculated through the equations in Section 3, employing a Gaussian distribution to quantify for every node the number of edges explored per tree level, E_l. This process is not followed in the first phase, where E_l trivially equals the connectivity of the root vertex. In [11], Kim and Leskovec demonstrate that the vertex degree in Kronecker matrix product-generated graphs such as the one used in the Graph500 Benchmark can be approximated through a lognormal distribution with the mean and standard deviation in Equations 18- 19.

$$\overline{d} = \ln\left((0.3604)^s\right) + 1.0661704 \cdot s \quad (18)$$

$$\sigma_d = 0.079313065 \cdot s \quad (19)$$

Whenever a point-to-point message is generated, the node attempts to insert it in the injection buffer of the directly-linked router. Following the *simple* implementation of the benchmark, the node is not able to generate new messages until all received messages have been consumed. New message generation is halted while an incoming message is being consumed or an outgoing message cannot be injected into the network (due to lack of space, or buffer in use).

After a node has delivered all the messages for the current stage, it dispatches a signal to all other nodes in the network. When all nodes have finished sending and receiving messages, they enter an all-reduce phase comprising a reduce (all nodes send to one) and a broadcast (one node sends to the rest). The node selected as host for the root vertex will act as the receiver for the messages in the former case and transmitter in the latter.

Following the all-reduce, the simulator determines if the graph has been wholly traversed (all the queries have been sent, upper bounded by twice the number of edges in the graph). If not, they re-enter the level start phase and proceed with another stage; otherwise, the execution ends.

5. MODEL VALIDATION

To validate our model, we have employed a two-phase approach: first, we have crosschecked the validity of the prediction from our equations; then, we have implemented our model in the FOGSim network simulator [4] and have analyzed the execution outcome for different configurations.

5.1. Validation of the equations for the number of explored edges per tree level

Since the impact of message aggregation is clearly defined by Equation 3, we compare the outcome of the equations for the mean ($\overline{E_l}$) and standard deviation (σ_{E_l}) of the number of explored edges per tree level against the values measured from a set of Graph500 executions with parameters different than those employed to obtain the equations.

Figure 7a displays the average number of explored edges $\overline{E_l}$ for all possible root vertices in a graph of *scale* $s = 22$ and *edgefactor* $f_e = 16$. Points correspond to the measured values, whereas

Prepared using cpeauth.cls
DOI: 10.1002/CPE.4231

(a) Mean. (b) Standard Deviation.

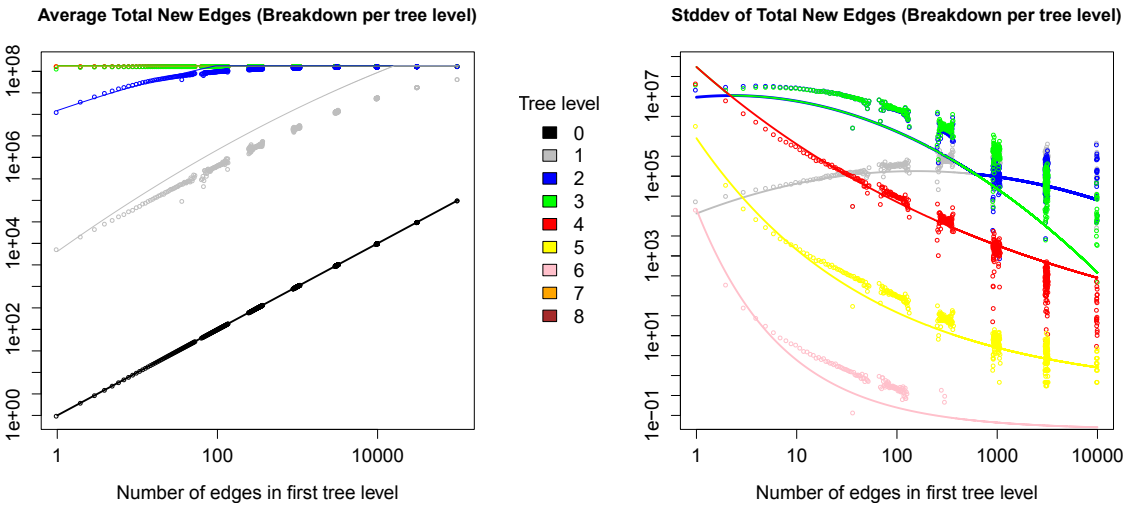


Figure 7. Validation of the model. Points correspond to measured average and standard deviation values from a real execution, lines correspond to the fittings from the model.

lines are the fittings obtained through our model. The fitting curves approximate clearly the observed behavior, following the same trends as the measurements for every stage of the execution. The model reproduces the staged behavior and replicates the dependence on the root degree, observing a similar proportion between the impact of each stage in the total amount of explored edges. From the second level $l = 1$ we observe that the model result is truncated for large root degrees when the maximum number of edges given by the *scale* and *edgefactor* are explored.

The dynamic range of the observed values is very large due to its logarithmic nature; this implies that any deviation in the prediction will incur in a very large absolute error. Still, the relative error of our model for this second tree level is lower than 18% in more than 90% of the cases. For the third tree level, which amounts the largest amount of communications for most root degrees, the model is still able to reproduce the same behavior with an average relative error of 12.5%. The total number of explored edges across the graph presents a relative error below 12%, which is corrected when the maximum value is reached and the edges in the last levels are truncated.

Figure 7b depicts in a similar fashion the measured points and fittings for the standard deviation. Deviation between the measured points and the model prediction is larger in this case; however, the dispersion for large root degree values is so large that any fitting must necessarily incur in substantial errors. The model curves nevertheless resemble the trend of the samples.

A similar analysis has been conducted for the mean and standard deviation upon graphs of *scale* $s = 18$ and *edgefactor* $f_e = 40$, with analogous results.

5.2. *Execution results from the network simulator*

We have run a battery of simulations in the FOGSim network simulator [4] with the Graph500 synthetic traffic model to evaluate its accuracy. We employ a Dragonfly network [15] with 72 computation nodes, and input-output-buffered routers with several virtual channels to avoid deadlock and mitigate Head-of-Line blocking. We use a simple model of a router with a 200-cycle pipeline. As a reference point, we have considered the specifications of the 40Gbps QDR/FDR10 InfiniBand (IB) switch in the Altamira supercomputer to calculate the network-related input parameters summarized in Table IV. Each point represents the average of 10 executions; they correspond to 10 different graph roots with different degree (the same set for all points), selected randomly according to the lognormal distribution.

Prepared using cpeauth.cls
DOI: 10.1002/CPE.4231

Table IV. Simulation parameters.

Parameter	**Value**
Number of nodes	72
Network topology	Dragonfly [15]
Router size	7 ports (h=2 global, p=2 injection, 3 local)
Group size	4 routers, 8 computing nodes
System size	9 groups
Latency	50/500 cycles (local/global links), 200 cycles (router)
Virtual Channels	2/1 (local/global input ports) for MIN, 4/2 for VAL, 3 injection buffers
Buffer size	16kB (local input per VC, output), 128kB (injection and global input per VC)
Packet size	4kB
Freq. speedup	2×
Switching	Virtual Cut-Through
Allocator	Iterative input-first separable allocator
Routing	Minimal

(a) Node capability. (b) Bandwidth.

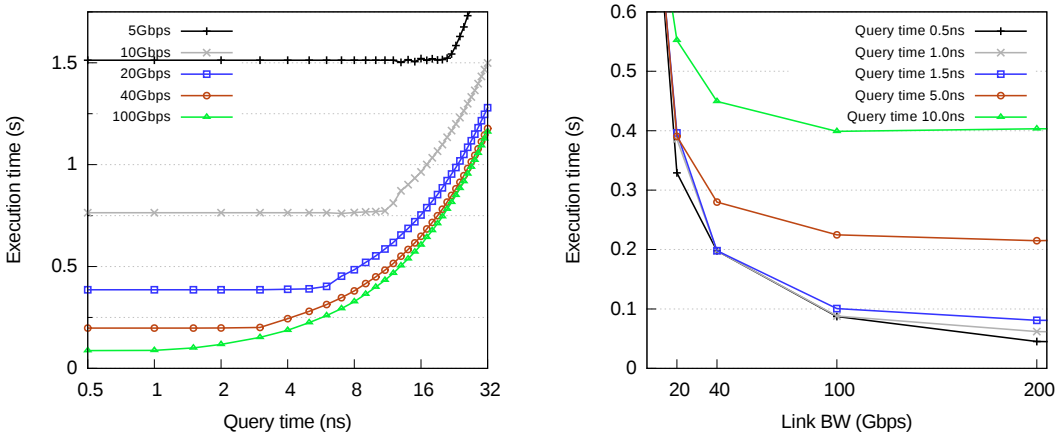


Figure 8. Execution time for a sweep in node computation capability and link bandwidth (BW), with a graph of scale 26 and edgefactor 16.

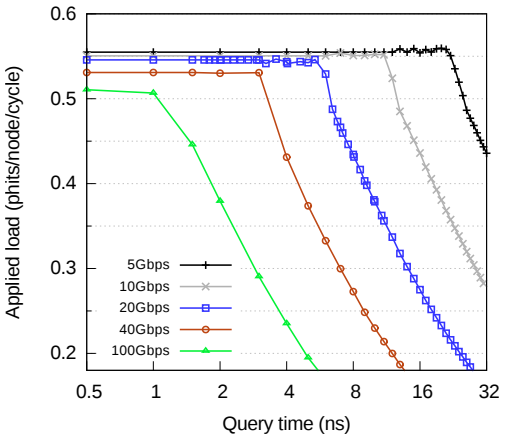


Figure 9. Network usage per node computation capability, under different link bandwidths. Results for a graph of scale 26 and edgefactor 16.

Prepared using cpeauth.cls
DOI: 10.1002/CPE.4231

Figure 8 illustrates the impact of the node computation capabilities and the link bandwidth (BW) on the execution time of one BFS execution, in a graph of scale $s = 26$ and edgefactor $f_e = 16$, with a coalescing size of $c_s = 256$. Figure 8a depicts the execution time in seconds for a sweep in the query computation time (t_q), for different link bandwidths. The curves for the execution time present two clearly different behaviors. For long query times, the execution time grows with the query computation time. This zone corresponds to CPU-bounded execution, where the node computational capabilities act as the bottleneck for the performance of the benchmark. However, for short query computation times the execution time stales, and the network becomes a bottleneck for the system performance. The frontier between these zones represents a balanced system. Interestingly, for a system with the characteristics of the Altamira supercomputer (query time $t_q = 1.5ns$ and 40Gbps of link bandwidth) the execution time is already hindered by the network limitations.

The impact of the network is restricted to the bandwidth and not the latency characteristics. A simulation employing links with a 10x increase in delay (omitted for the sake of brevity) rendered negligible differences from the displayed curves. This behavior is related to the nature of the communications of the benchmark, which do not present any interdependencies between messages within the same tree level and therefore favor higher message dispatch rates over lower latencies.

Figure 8b renders the execution time in seconds upon the link bandwidth, for different query computation times (and thus, different node computation capabilities). For a node with a query computation time $t_q = 10ns$ (similar to the ARM CPU of the Mont-Blanc prototype) an increase in link BW beyond 40Gbps is close to ineffective, but for a query time below 1.5ns an improvement from 40Gbps to 100Gbps halves the execution time. Performance in the zone limited by the network is not completely proportional to the link BW, because the model does not only consider the point-to-point communications but also the synchronization phases (through the end-of-level signals and the all-reduce operations).

Figure 9 portrays the network usage for the same curves in Figure 8a. The half-duplex nature of the program (where the nodes cannot generate new messages when they are consuming an incoming packet) forces that for all link bandwidths the curves saturate at close to 0.5 phits/node/cycle. Out of that region, the query time determines the message injection rate; for a given query computation time, a higher bandwidth gives a lower injection rate.

Each execution of the simulator has required less than 100MB of memory in a single process; as a reference point, the capture of the behavior for a scale $s = 23$ (8 times smaller than the scale $s = 26$ employed in our simulations) and 64 processes involves a trace of 600MB. Furthermore, trace size scales almost linearly with the size of the graph and the number of processes, making unfeasible to evaluate larger cases. By contrast, our model only requires a longer execution to perform the analysis of greater graph sizes.

6. RELATED WORK

Graph500 was introduced in 2010 as a benchmark data intensive applications. Beamer *et al.* characterize the memory requirements and locality of the benchmark in [16], but they do not study the impact of the network and its utilization. Previous works from Anghel *et al.* [17] [18] and Fuentes *et al.* [7] provide a comprehensive characterization of the communications in the *simple* implementation of the benchmark. Our previous work in [3] identified the strong dependence between root vertex connectivity and BFS execution and provided a first model of Graph500 traffic, which has been extended in Section 3.

The distribution of the vertex degree in graphs generated with the R-MAT algorithm can be modeled through a power-law or lognormal distribution [8, 11, 10]. Groër *et al.* [19] provide a more accurate model as a series expansion from normal distributions, and Seshadhri *et al.* [20] simplify it as a combination of a lognormal and an exponential tail distribution. The distribution in our model can be readily replaced by either of these approaches should a more precise distribution be needed.

Network simulators constitute a useful tool for network architects in the design and evaluation of new systems. Simulators are commonly based on full-system simulation, trace-driven execution or synthetic traffic patterns. Full system simulators such as Gem5 [21] present high computational

Prepared using cpeauth.cls
DOI: 10.1002/CPE.4231

and memory requirements; an alternative is to replace the application by a dwarf [22], or a skeleton restricted to the most relevant sections for the network evaluation, as done in the SST simulator [23]. Such options are not feasible for the Graph500 benchmark, because the core interest of the network simulation requires the graph traversal, and presents similar memory restrictions to the simulation of the whole application. Trace-driven simulators, such as Dimemas [24], Netrace [25] or the VEF framework [26], fail to accurately represent the dependencies in the execution. Additionally, the size of traces from data-intensive applications is too big and increases with the number of processes.

Synthetic traffic models, as the one presented in Section 3, have smaller computational and memory requirements than the previous alternatives while retaining the core characteristics of the workload they represent. Synthetic traffic models have traditionally consisted of permutations to determine the destination or set of destinations for the messages from a given node, but this does not fit the behavior of BigData applications. SynFull [27] generates synthetic network traffic which preserves the characteristics (temporality, destinations, volume, etc.) from executions of a real application. It relies on Markov Chains to model the behavior of multiple phases of an application. However, it is focused on Networks-on-Chip only, modeling memory accesses and their associated coherence traffic, which is not appropriate for system-level interconnects. [28] introduces a 3-tuple statistical model to generate synthetic traffic and [29] presents synthetic traffic traces which model the communications in a video application, but they are both oriented to NoC traffic. An example of the use of a traffic model and its application to real systems has been presented in Section 5.2, where the node computing performance is related to maximum application speedups derived from improvements in the network.

7. CONCLUSIONS

Current evaluations of BigData workloads consist mostly of full-system simulations of the real applications, or rely on the use of traces. Both options limit severely the size and detail of the network that can be investigated via simulation - which confers observability otherwise unattainable. Here we have introduced a novel computationally non-intensive synthetic traffic model of the most scalable implementation of the Graph500 benchmark. We have analyzed the distribution of the benchmark communications in stages and its relation with the number of explored edges per tree level. Furthermore, we have identified a strong connection to the degree of the vertex selected as root of the tree.

We have modeled the benchmark behavior as a set of stages of point-to-point messages separated by all-reduce collective operations. The number of messages is defined as a linear model of the benchmark parameters (*scale*, *edgefactor*) for each stage within the BFS computation (*tree level*). Using an empirical characterization of actual benchmark executions for different graph parameters, we have defined a set of equations to compute the mean and standard deviation of a normal distribution that determines the number of edges per tree level for any given tree root degree; the degree of the root vertex is decided randomly following a lognormal distribution.

We have provided an implementation of our model for the FOGSim network simulator, which can be handily adapted for other network simulators. Our implementation features high scalability with the size of the graph and the number of nodes employed. We also provide a short list of system architectures and its associated computation time as a reference value for the input parameter of the model. Following the network parameters in the simulator that correspond to a reference InfiniBand router architecture, we perform a set of simulations and provide a figure with the model execution time for different node capabilities and link bandwidths. Our results demonstrate that the execution time of the model presents two distinct behaviors: a CPU-bounded region where execution time grows with query computation time, and a flat region when the node computation capability exceeds the maximum injection rate of the network. Maximum performance in network-limited regions does not only account for point-to-point communications but also synchronization phases.

ACKNOWLEDGEMENT

DOI: 10.1002/CPE.4231

The authors would like to thank the European HiPEAC Network of Excellence for partially funding this work through a Collaboration Grant. They would like to thank as well Cristóbal Camarero for his help. This work has been supported by the Spanish Ministry of Education, FPU grants FPU13/00337 and FPU14/02253, the Spanish Ministry of Economy, Industry and Competitiveness under contract TIN2016-76635-C2-2-R (AEI/FEDER, UE), and by the Mont-Blanc project. The Mont-Blanc project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 671697. Santander Supercomputacion support group from the University of Cantabria provided access to the Altamira Supercomputer at the Institute of Physics of Cantabria (IFCA-CSIC).

REFERENCES

1. Graph500 benchmark May 2016. URL `http://www.graph500.org/`.
2. Murphy RC, Wheeler KB, Barrett BW, Ang JA. Introducing the Graph 500. *Cray User's Group (CUG)* 2010; .
3. Fuentes P, Vallejo E, Bosque JL, Beivide R, Anghel A, Rodríguez G, Gusat M, , Minkenberg C. Synthetic traffic model of the Graph500 communications. *Proceedings of the 16th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP)*, 2016.
4. García M, Fuentes P, Odriozola M, Vallejo E, Beivide R. FOGSim Interconnection Network Simulator. University of Cantabria 2014. URL `http://fuentesp.github.io/fogsim/`.
5. Suzumura T, Ueno K, Sato H, Fujisawa K, Matsuoka S. Performance characteristics of Graph500 on large-scale distributed environment. *International Symposium on Workload Characterization (IISWC)*, IEEE, 2011; 149–158.
6. Ueno K, Suzumura T. Highly scalable graph search for the Graph500 benchmark. *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*, HPDC '12, ACM, ACM: New York, NY, USA, 2012; 149–160, doi:10.1145/2287076.2287104. Doi:10.1145/2287076.2287104.
7. Fuentes P, Bosque JL, Beivide R, Valero M, Minkenberg C. Characterizing the communication demands of the Graph500 benchmark on a commodity cluster. *Int. Symposium on Big Data Computing*, 2014; 83–89.
8. Chakrabarti D, Zhan Y, Faloutsos C. R-MAT: A recursive model for graph mining. *Proceedings of the 2004 SIAM International Conference on Data Mining*; 442–446, doi:10.1137/1.9781611972740.43.
9. Watts DJ, Strogatz SH. Collective dynamics of 'small-world' networks. *Nature* 1998; **393**(6684):440–442.
10. Leskovec J, Chakrabarti D, Kleinberg J, Faloutsos C, Ghahramani Z. Kronecker graphs: An approach to modeling networks. *The Journal of Machine Learning Research* 2010; **11**:985–1042.
11. Kim M, Leskovec J. Multiplicative attribute graph model of real-world networks. *Algorithms and Models for the Web-Graph*. Springer, 2010; 62–73.
12. Papoulis A. Bernoulli trials. *Random Variables, and Stochastic Processes*, McGraw-Hill, New York, 1990; 57–63.
13. Chambers J, Hastie T. *Statistical Models in S*. Wadsworth & Brooks/Cole, 1992.
14. Rajovic N, Rico A, Mantovani F, Ruiz D, Vilarrubi J, Gomez C, Nieto D, Servat H, Martorell X, Labarta J, *et al.*. The Mont-Blanc prototype: An alternative approach for HPC systems. *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2016.
15. Kim J, Dally W, Scott S, Abts D. Technology-driven, highly-scalable dragonfly topology. *ISCA'08: 35th International Symposium on Computer Architecture*, IEEE Computer Society, 2008; 77–88.
16. Beamer S, Asanovic K, Patterson D. Locality exists in graph processing: Workload characterization on an Ivy Bridge server. *Workload Characterization (IISWC), 2015 IEEE International Symposium on*, 2015; 56–65.
17. Anghel A, Rodriguez G, Prisacari B. The importance and characteristics of communication in high performance data analytics. *Workload Characterization (IISWC), 2014 IEEE International Symposium on*, IEEE, 2014; 80–81.
18. Anghel A, Rodriguez G, Prisacari B, Minkenberg C, Dittmann G. Quantifying communication in graph analytics. *High Performance Computing*, Springer, 2015; 472–487.
19. Groër C, Sullivan BD, Poole S. A mathematical analysis of the R-MAT random graph generator. *Networks* 2011; **58**(3):159–170.
20. Seshadhri C, Pinar A, Kolda TG. An in-depth study of stochastic Kronecker graphs. *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, IEEE, 2011; 587–596.
21. Binkert N, Beckmann B, Black G, Reinhardt SK, Saidi A, Basu A, Hestness J, Hower DR, Krishna T, Sardashti S, *et al.*. The Gem5 simulator. *SIGARCH Comput. Archit. News* Aug 2011; **39**(2):1–7, doi:10.1145/2024716.2024718.
22. Asanovic K, Bodik R, Catanzaro BC, Gebis JJ, Husbands P, Keutzer K, Patterson DA, Plishker WL, Shalf J, Williams SW, *et al.*. The landscape of parallel computing research: A view from berkeley. *Technical Report UCB/EECS-2006-183*, EECS Department, University of California, Berkeley Dec 2006.
23. Rodrigues AF, Hemmert KS, Barrett BW, Kersey C, Oldfield R, Weston M, Risen R, Cook J, Rosenfeld P, CooperBalls E, *et al.*. The structural simulation toolkit. *SIGMETRICS Perf. Eval. Rev.* Mar 2011; **38**(4):37–42.
24. Badia RM, Labarta J, Gimenez J, Escale F. Dimemas: Predicting mpi applications behavior in grid environments. *Workshop on Grid Applications and Programming Tools (GGF8)*, vol. 86, 2003; 52–62.
25. Hestness J, Grot B, Keckler SW. Netrace: Dependency-driven trace-based network-on-chip simulation. *Proceedings of the Third International Workshop on Network on Chip Architectures*, NoCArc '10, ACM: New York, NY, USA, 2010; 31–36, doi:10.1145/1921249.1921258. URL `http://doi.acm.org/10.1145/1921249.1921258`.
26. Andújar FJ, Villar JA, Sánchez JL, Alfaro FJ, Escudero-Sahuquillo J. VEF traces: A framework for modelling MPI traffic in interconnection network simulators. *International Conference on Cluster Computing*, 2015; 841–848.
27. Badr M, Jerger NE. SynFull: Synthetic traffic models capturing cache coherent behaviour. *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, 2014; 109–120, doi:10.1109/ISCA.2014.6853236.
28. Soteriou V, Wang H, Peh L. A statistical traffic model for on-chip interconnection networks. *14th IEEE International Symposium on Modeling, Analysis, and Simulation*, 2006; 104–116, doi:10.1109/MASCOTS.2006.9.
29. Varatkar GV, Marculescu R. On-chip traffic modeling and synthesis for MPEG-2 video applications. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* Jan 2004; **12**(1):108–119.